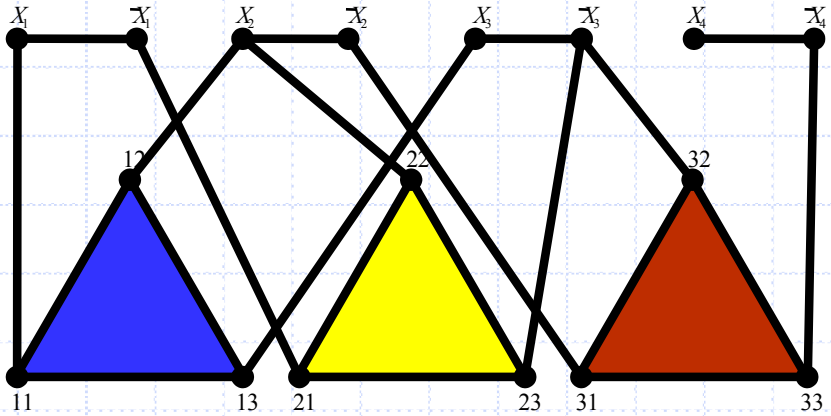
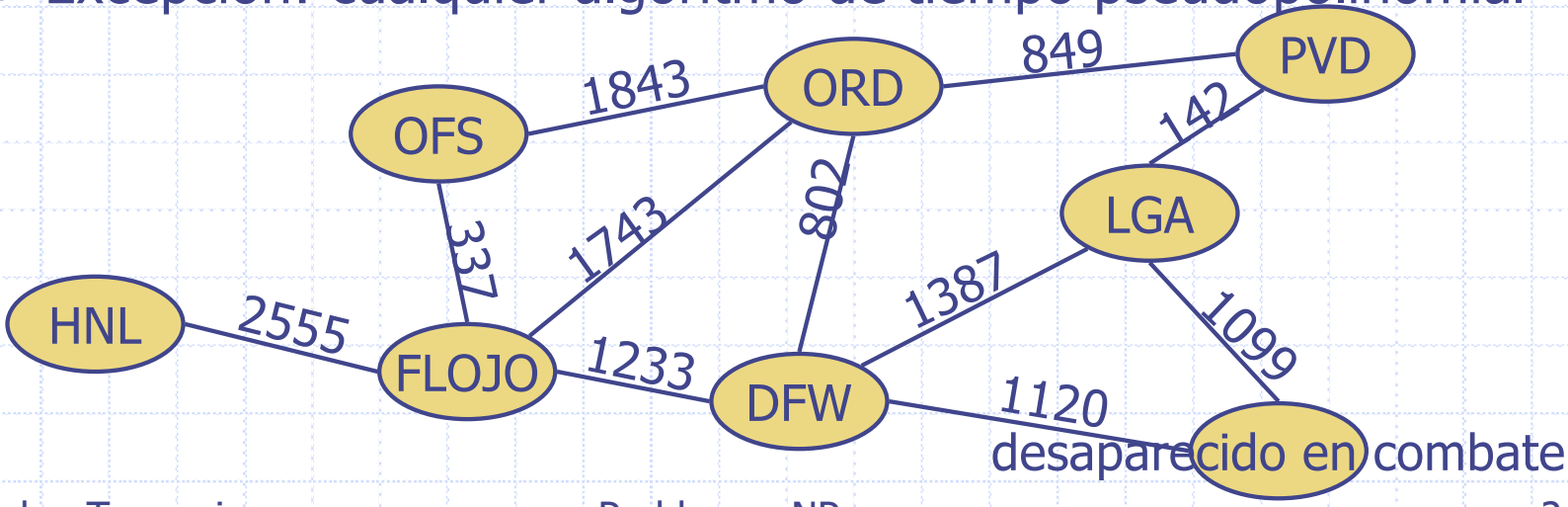
Presentación para usar con el libro de texto Diseño y aplicaciones de algoritmos , de MT Goodrich y R. Tamassia, Wiley, 2015

Problemas NP



Tiempo de ejecución revisado

- Tamaño de entrada, ***n***
  - Para ser exactos, sea ***n*** el número de **bits** en una codificación no unaria de la entrada
- Todos los algoritmos de tiempo polinomial estudiados hasta ahora en este curso se ejecutan en tiempo polinomial utilizando esta definición de tamaño de entrada.
  - Excepción: cualquier algoritmo de tiempo pseudopolinomial



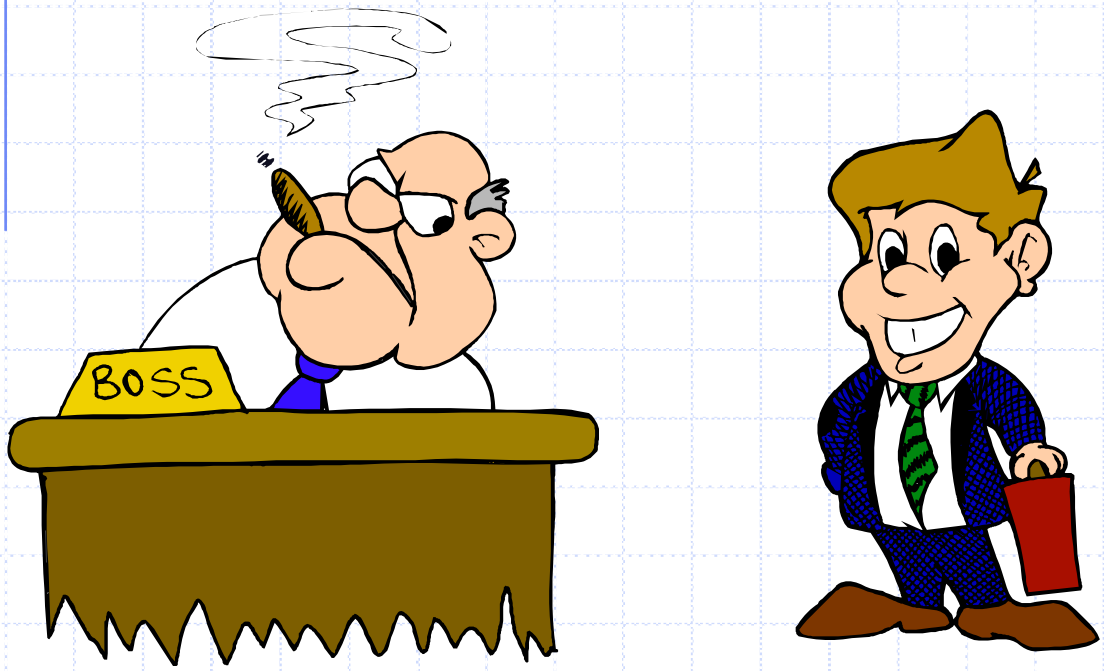
Lidiar con problemas difíciles

- Qué hacer cuando encontramos un problema que parece difícil...

No pude encontrar un algoritmo de tiempo polinomial;
Supongo que soy demasiado tonto.

Lidiar con problemas difíciles

- A veces podemos demostrar un límite inferior fuerte... (pero no normalmente)



No pude encontrar un algoritmo de tiempo polinomial,
¡Porque no existe tal algoritmo!

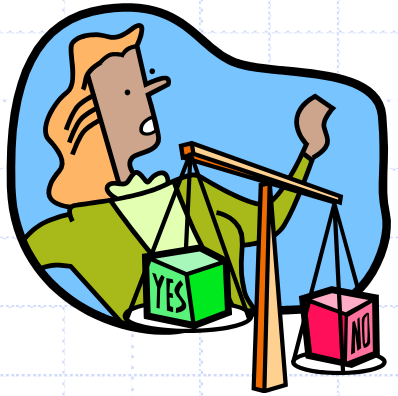
Lidiar con problemas difíciles

- La integridad NP nos permite mostrar colectivamente que un problema es difícil.

No pude encontrar un algoritmo de tiempo polinomial,
pero tampoco podrían hacerlo todas estas otras personas inteligentes.

(caricatura inspirada en [Garey-Johnson, 79])

Problemas de decisión en tiempo polinomial

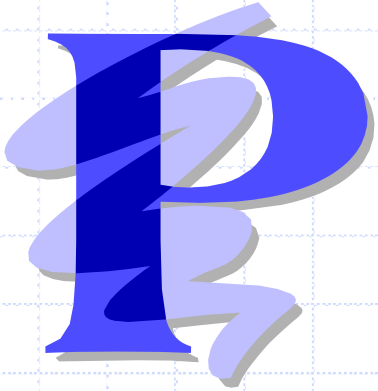


- Para simplificar la noción de " dureza " , nos centraremos en lo siguiente:
  - El tiempo polinómico como límite para la eficiencia
  - Problemas de decisión: la salida es 1 o 0 (" sí " o " no ")
    - Ejemplos:
    - ¿Tiene un gráfico G dado un recorrido de Euler?
    - ¿Un texto T contiene un patrón P?
    - ¿Tiene una instancia de 0/1 Knapsack una solución con un beneficio de al menos K?
    - ¿Tiene un gráfico G un MST con peso como máximo K?

Problemas y lenguajes

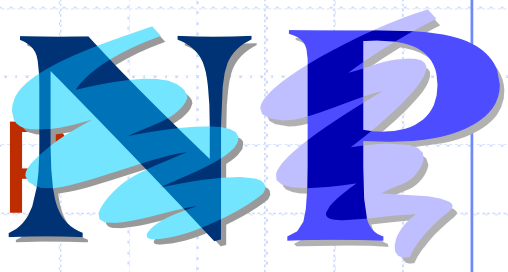
- Un **lenguaje** L es un conjunto de cadenas definidas sobre algún alfabeto Σ
- Cada algoritmo de decisión A define un lenguaje L
  - L es el conjunto que consta de cada cadena x tal que A genera " sí " en la entrada x.
  - Decimos " A **acepta** x " en este caso
    - Ejemplo:
    - Si A determina si un gráfico dado G tiene o no un recorrido de Euler, entonces el lenguaje L para A son todos los gráficos con recorridos de Euler.

La clase de complejidad P



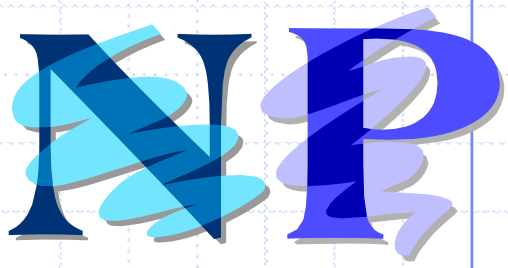
- Una **clase de complejidad** es una colección de idiomas.
- P es la clase de complejidad que consta de todos los lenguajes aceptados por algoritmos **de tiempo polinomial.**
- Para cada lenguaje L en P existe un algoritmo de decisión en tiempo polinómico A para L.
  - Si n=|x|, para x en L, entonces A se ejecuta en p(n) tiempo en la entrada x.
  - La función p(n) es algún polinomio

La clase de complejidad NP



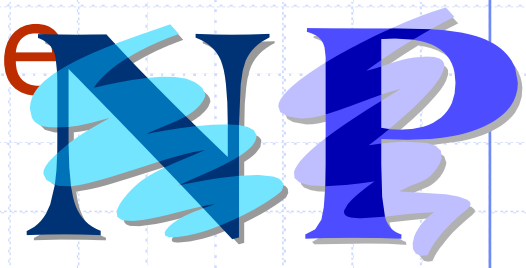
- Decimos que un algoritmo es no determinista si utiliza la siguiente operación:
  - Elegir(b): elige un poco b
  - Se puede usar para elegir una cadena completa y (con opciones |y|)
- Decimos que un algoritmo no determinista A **acepta** una cadena x si existe alguna secuencia de operaciones de elección que hace que A genere " sí " en la entrada x.
- NP es la clase de complejidad que consta de todos los lenguajes aceptados por el método **no determinista de tiempo polinómico.** algoritmos.

ejemplo de NP



- Problema: decidir si una gráfica tiene un MST de peso K
- Algoritmo:
  1. Elija de forma no determinista un conjunto T de n-1 aristas
  2. Pruebe que T forma un árbol de expansión
  3. Prueba que T tiene peso como máximo K
- Análisis: las pruebas requieren un tiempo O (n+m), por lo que este algoritmo se ejecuta en tiempo polinómico.

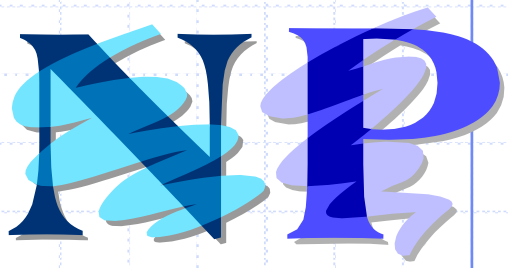
La definición alternativa de clase de complejidad NP



- Decimos que un algoritmo B **verifica** la aceptación de un lenguaje L si y sólo si, para cualquier x en L, existe un certificado y tal que B genera " sí " en la entrada (x,y).
- NP es la clase de complejidad que consta de todos los lenguajes verificados mediante algoritmos **de tiempo polinomial** .

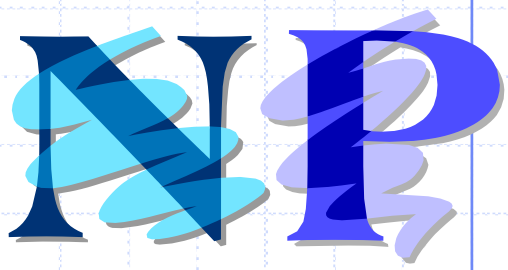
- Sabemos: P es un subconjunto de NP.
- Pregunta abierta importante: ¿P=NP?
- La mayoría de los investigadores creen que P y NP son diferentes.

Ejemplo de NP (2)



- Problema: decidir si una gráfica tiene un MST de peso K
- Algoritmo de verificación:
  1. Utilice como certificado, y, un conjunto T de n-1 aristas
  2. Pruebe que T forma un árbol de expansión
  3. Prueba que T tiene peso como máximo K
- Análisis: la verificación lleva un tiempo O (n + m), por lo que este algoritmo se ejecuta en tiempo polinómico.

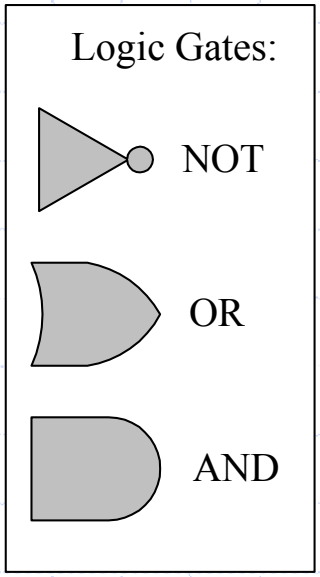
Equivalencia de las dos definiciones

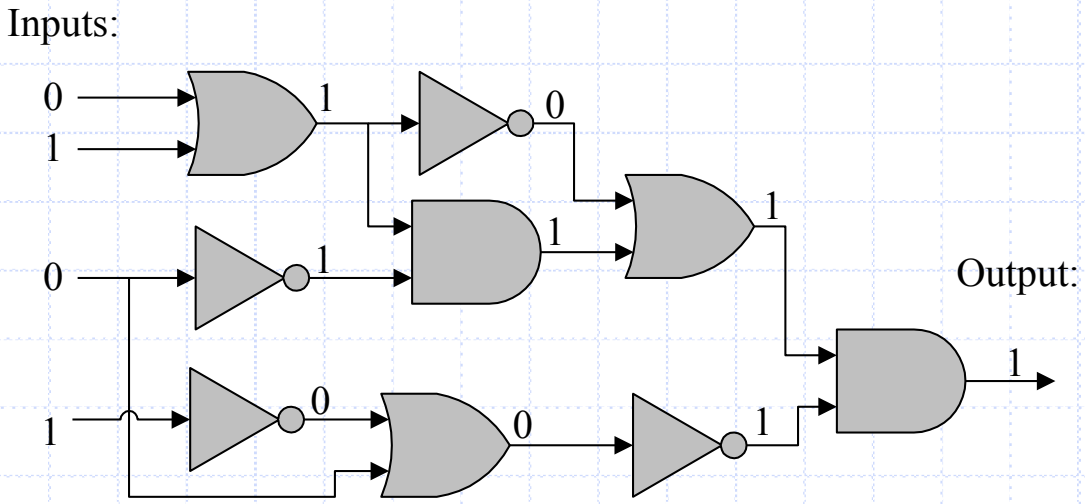


- Supongamos que A es un algoritmo no determinista.
  - Sea y un certificado que consta de todos los resultados de los pasos de elección que utiliza A.
  - Podemos crear un algoritmo de verificación que use y en lugar de los pasos de elección de A.
  - Si A acepta en x, entonces hay un certificado y que nos permite verificar esto (es decir, los pasos de elección que A realizó)
  - Si A se ejecuta en tiempo polinómico, también lo hace este algoritmo de verificación.
- Supongamos que B es un algoritmo de verificación.
  - Elija de forma no determinista un certificado y
  - Ejecute B en y
  - Si B se ejecuta en tiempo polinómico, también lo hace este algoritmo no determinista.

Un problema interesante

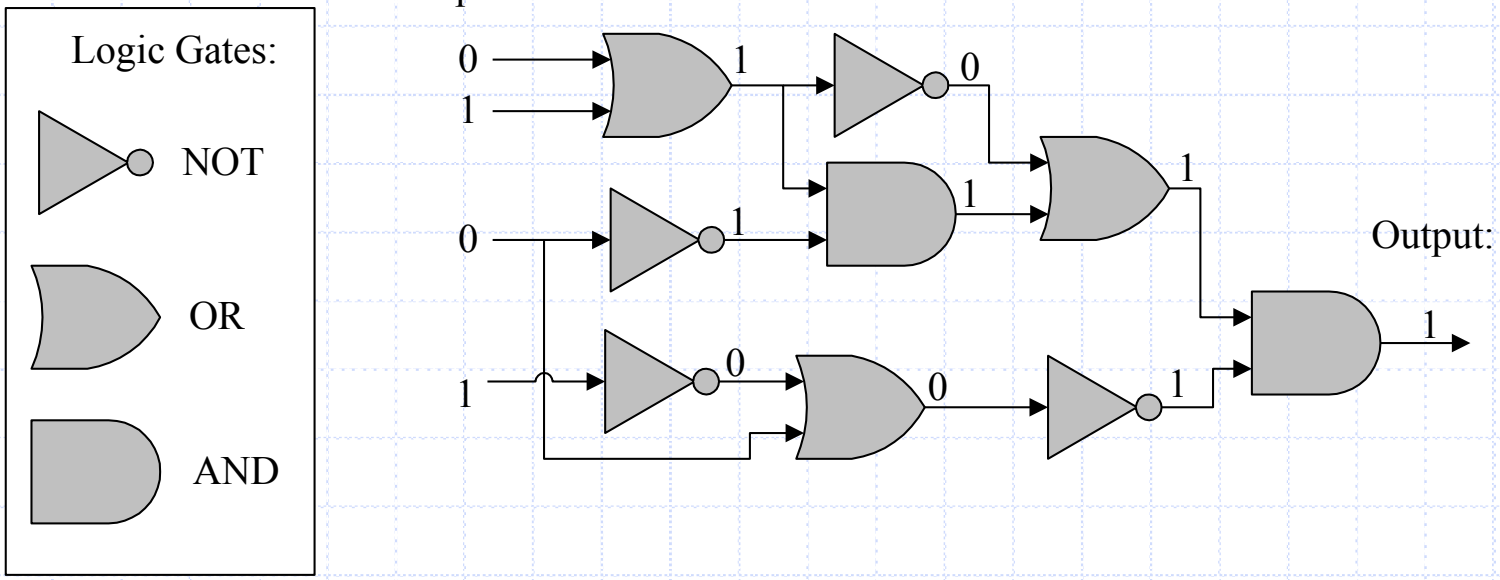
- Un circuito booleano es un circuito de puertas Y, O y NO; El problema de CIRCUIT-SAT es determinar si hay una asignación de 0 's y 1 's a las entradas de un circuito para que el circuito emita 1.





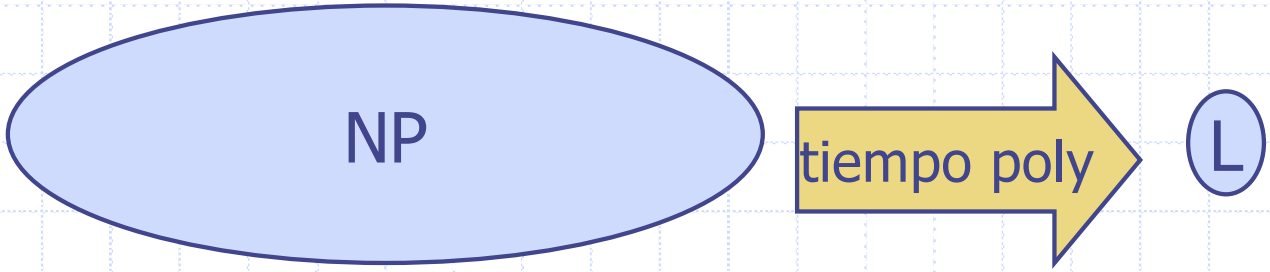
CIRCUITO-SAT está en NP

- Elija de forma no determinista un conjunto de entradas y el resultado de cada puerta, luego pruebe las E/S de cada puerta.



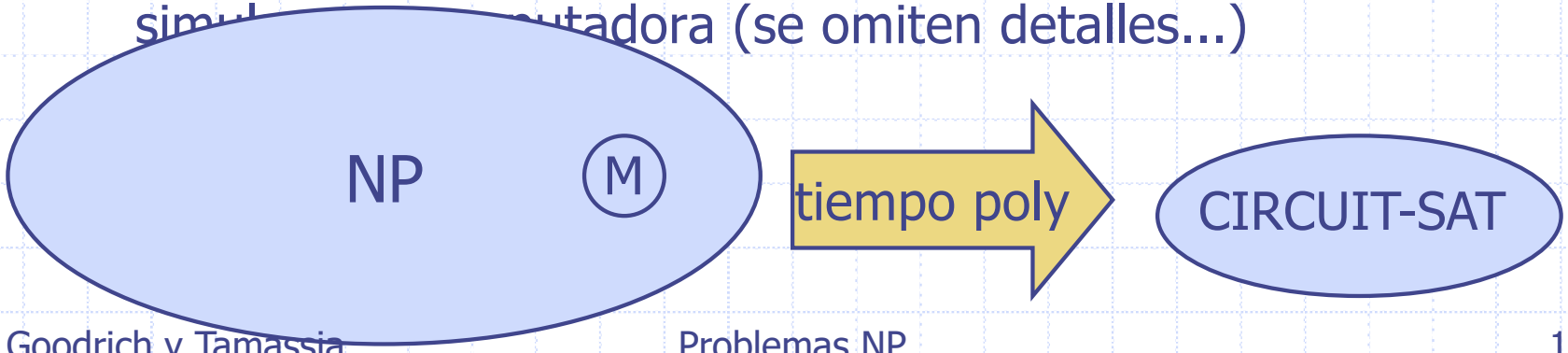
NP-integridad

- Un problema (lenguaje) L es **NP-difícil** si cada problema en NP se puede reducir a L en tiempo polinomial.
- Es decir, para cada lenguaje M en NP, podemos tomar una entrada x para M, **transformarla** en tiempo polinómico en una entrada x ' para L tal que x esté en M si y sólo si x ' está en L.
- L es **NP-completo** si está en NP y es NP-duro.



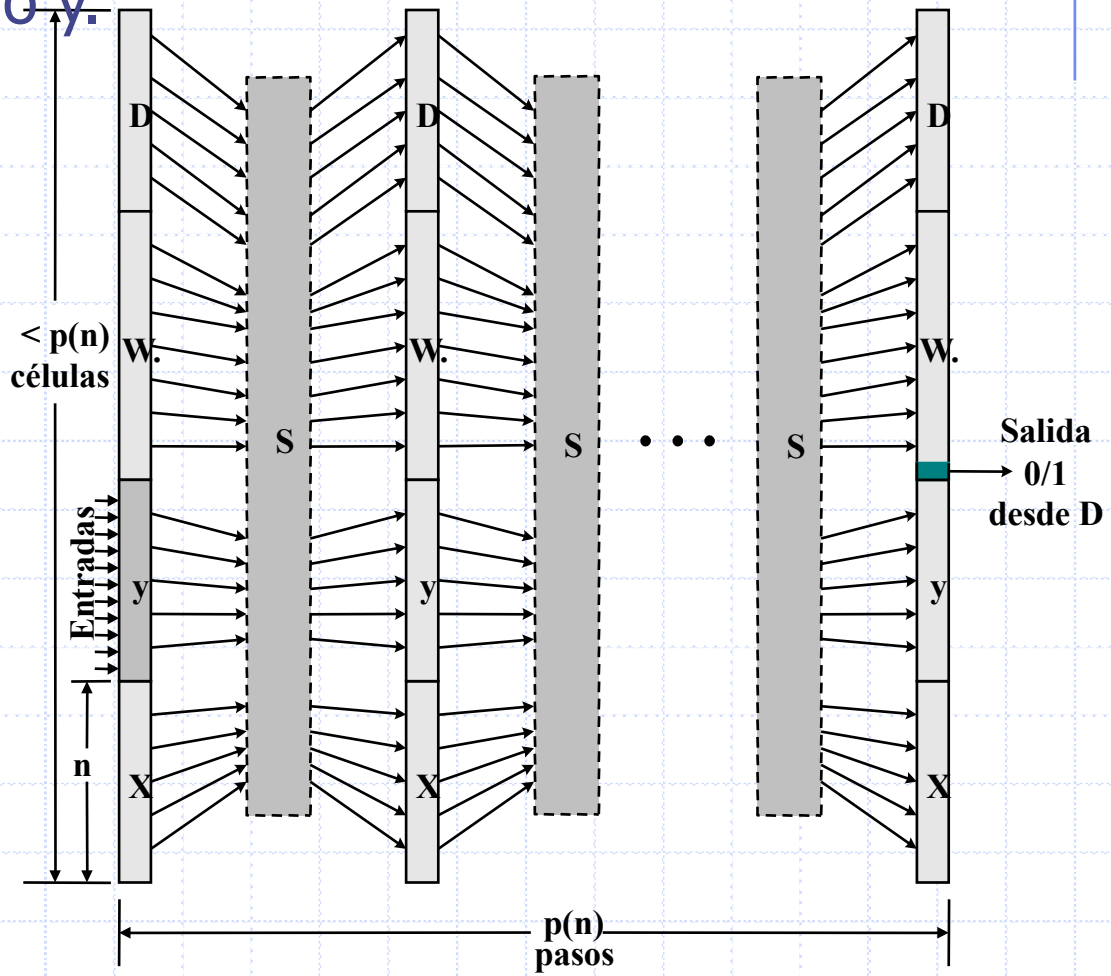
Teorema de Cook-Levin

- CIRCUIT-SAT es NP-completo.
  - Ya mostramos que está en NP.
- Para demostrar que es NP-duro, tenemos que demostrar que todos los idiomas en NP pueden reducirse a él.
  - Sea M en NP y sea x una entrada para M.
  - Sea y un certificado que nos permite verificar la pertenencia a M en tiempo polinómico, p(n), mediante algún algoritmo D.
  - Sea S un circuito de tamaño como máximo $O(p(n)^2)$ que simul[illegible]utadora (se omiten detalles...)

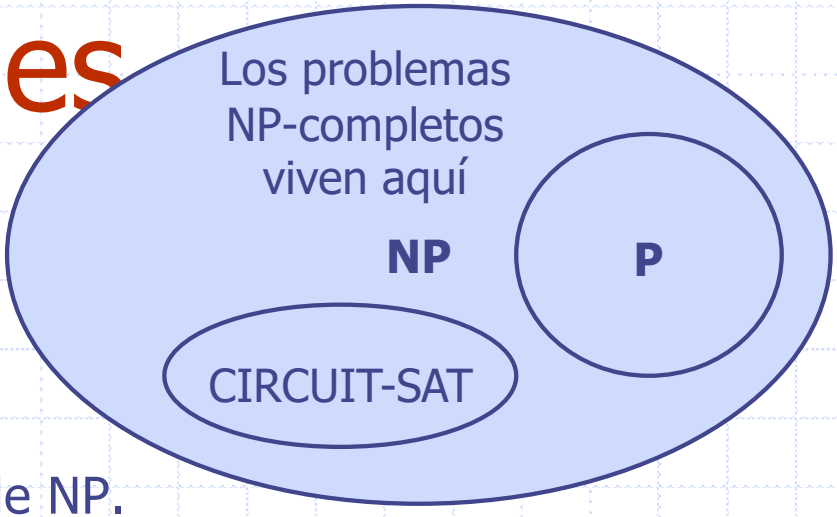


Prueba Cook-Levin

- Podemos construir un circuito que simule la verificación de la pertenencia de x a M usando y.
  - Sea W el almacenamiento de trabajo para D (incluidos los registros, como el contador de programa); dejemos que D se proporcione en RAM " código de máquina". "
  - Simule p(n) pasos de D replicando el circuito S para cada paso de D. Solo ingrese: y.
  - El circuito es satisfactorio si y sólo si x es aceptado por D con algún certificado y
  - El tamaño total sigue siendo polinomio: $O(p(n)^3)$.



Algunas reflexiones sobre P y NP



- Creencia: P es un subconjunto propio de NP.
- Implicación: los problemas NP-completos son los más difíciles en NP.
- Por qué: porque si pudiéramos resolver un problema NP completo en tiempo polinomial, podríamos resolver todos los problemas en NP en tiempo polinomial.
- Es decir, si un problema NP-completo se puede resolver en tiempo polinómico, entonces P=NP.
- Dado que muchas personas han intentado sin éxito encontrar soluciones en tiempo polinomial a problemas NP-completos, demostrar que su problema es NP-completo equivale a demostrar que muchas personas inteligentes han trabajado en su problema y no han encontrado ningún algoritmo de tiempo polinomial.